Vol.2015-SE-189 No.15
2015/7/23

不確かさを包容した開発プロセスとその支援環境 *iArch-U*

深町　拓也[1,a)]　鵜林　尚靖[1,b)]　細合　晋太郎[1,c)]　亀井　靖高[1,d)]

概要：本論文では，不確かさが存在しても設計，プログラミング，テストが継続可能な開発プロセスを提案する．我々は従来より不確かさを記述するためのインタフェース機構 *Archface-U* とそのコンパイラを提供してきた．本論文では，不確かさが発生する状況，それに対応するためのシナリオを例示し，開発プロセスの観点から我々のアプローチの有効性を示す．また，我々が開発を進めている開発環境 *iArch-U* の諸機能がこの開発プロセスのどこをサポートするのかを具体的に示す．

キーワード：不確かさ，Partial Model，統合開発環境

1. はじめに

ソフトウェア開発において，「不確かさ」は，絶えず変動し続けるビジネス環境や，ユーザの要求，システム運用などによって発生する．この不確かさは要求分析，設計，実装，テストといった様々なソフトウェア開発工程で現れる可能性がある．従来，開発者はこのようなソフトウェア開発における不確かさをリスク管理の一つとして扱い，リスク管理表や，仕様書などにメモとして記載するなどの対処をしてきた (図 1)．しかし，このような対処法では実装やテストの段階において何が不確かなのかがわからなくなることが多く，バグやコードの煩雑化の原因となることがある．そこで，我々は従来より，不確かさを適切に記述し，不確かさをコード上に抱えていても，実装を進めることができるインターフェース機構として *Archface-U* とそのコンパイラを提案してきた [7]．

本論文では，このインターフェースを用いて不確かさが存在していても設計，実装，およびテストが継続可能な開発プロセスを提案する．不確かさが発生する状況，それに対応するためのシナリオを例示し，開発プロセスの観点から我々が提供してきた *Archface-U* の有用性を示す．また，*Archface-U* の開発環境である *iArch-U* の機能がこの開発プロセスのどの部分をサポートするのかを具体的に示す．

本論文では，2 章では，関連研究をもとに *Archface-U* で扱う不確かさがどの種類の不確かさかを明らかにする．3 章では，本開発プロセスで用いるインターフェース *Archface-U* と，Partial Model について説明し，本開発プロセスについて述べる．4 章では，*Archface-U* の開発環境である *iArch-U* を中心とした本開発プロセスを支援するツールを説明する．5 章では，不確かさを包容した開発プロセスを適用したシナリオを説明する．最後に，6 章で本研究のまとめを述べる．

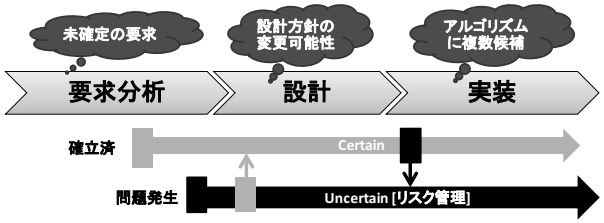

図 1　ソフトウェア開発における不確かさとその従来の対処

2. 関連研究

本章では，関連研究をもとにソフトウェア開発における不確かさが，どのように分類されるのかを述べる．後に，我々が提供してきた不確かさを扱うインターフェース機構 *Archface-U* が，どの分類の不確かさを扱うのかを示す．

2.1 不確かさの分類

Diego らは，不確かさについて，「場所 (*Location*)」，「レベル (*Level*)」，「性質 (*Nature*)」の 3 つの観点から分類をすることができるとしている [5](図 2)．以下では，*Archface-U* の扱う不確かさがどのようなものかを示すため，Diego らの不確かさの分類について簡単に説明をする．

2.1.1 場所による分類

不確かさの「場所」とは，モデルの中でその不確かさが

[1] 九州大学
Kyushu University
[a)] fukamachi@posl.ait.kyushu-u.ac.jp
[b)] ubayashi@ait.kyushu-u.ac.jp
[c)] hosoai@qito.kyushu-u.ac.jp
[d)] kamei@ait.kyushu-u.ac.jp

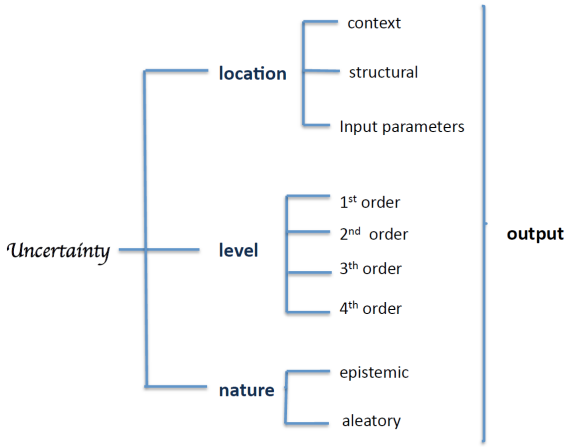

図 2　Diego らによる不確かさの分類 [5]

どこに現れるのかを指している．そして，不確かさは以下の 3 つの場所に現れる可能性があるとしている．

- コンテキスト：モデル化される情報に現れる不確かさ．
- モデル構造：モデル自体の構造に現れる不確かさ．
- 入力パラメータ：モデルに入力するパラメータにおける不確かさ．

2.1.2　レベルによる分類

不確かさの「レベル」とは，確定している知識をレベル 0 とし，総合的に全く何もわからない状態をレベル 4 として分類したものである．このレベルは以下のように説明できるとしている．

- レベル **0**：確定している知識．
- レベル **1**：知識が不足している状態．すなわち，既知の不確かさ．
- レベル **2**：知識が不足している上，その認知も不足している状態．すなわち，未知の不確かさ．
- レベル **3**：不確かさの認知が不足していることを認識するプロセスが不足している状態．つまり，認知をしない限りは何も行動を起こさないような不確かさ．
- レベル **4**：超不確実性．不確かさの次元に対する不確かさ．

2.1.3　性質による分類

不確かさの「性質」は，知識の不完全性によるものと，現象の固有変動性によるものとに以下のように二分できるとしている．

- 認識的：十分なデータや知識理解ができていないため発生する不確かさ．
- 偶発的：ランダム性のあるイベント固有の不確かさ．

2.2　本研究における「不確かさ」

本研究では，現在明らかになっている不確かさを記述し，明記することで不確かさが含んでいても開発を継続できることを目的とする．そのため，本研究では，2.1 節の不確かさの分類のうち以下の不確かさを扱う．

- 場所による分類 − コンテキスト，モデル構造，入力パラメータ
- レベルによる分類 − レベル 1，レベル 0
- 性質による分類 − 認識的

本研究は認識することができるレベル 1 の不確かさを扱う．また，*Archface-U* では「不確かではない」という状態を記述することもできるため，レベル 0 の不確かさ (正確には確定事項) も扱う．

3.　不確かさを包容した開発プロセス

本章では，不確かさを包容した開発プロセスを説明する．まず，その開発プロセスに用いられる以下の 2 つの既存研究について説明する．一つは，Famelis らによって提案された不確かさを包容したモデル，Partial Model [2] である．もう一つは，我々が提案する不確かさを包容するインターフェース機構 *Archface-U* [7] である．

後に，それらの既存研究を用いてどのように不確かさを包容した開発プロセスを実現するかを説明する．

3.1　Partial Model

Famelis らは設計初期段階において起こりうる，「いくつかの設計候補がありそれらの中からどれを適用するかが定まっていない」という不確かさを扱っている．Partial Model とは，複数の設計候補を一つのモデルにまとめたものである．これは，どの設計候補が選ばれるかわからないという不確かさを含んでいる．Partial Model で扱う不確かさは，Diego らによる不確かさの分類によると本研究と同じである．しかし，*Archface-U* は主に実装とテスト，Partial Model は設計モデルを対象とするという違いがある．

図 3 では，ある P2P ファイル共有システムにおける 6 つの設計候補 (a-f) をまとめた Partial Model (g) を表している．Partial Model では，複数のモデル全てにおいて共通しているエッジ，およびノードを実線で表し，それ以外を破線で表現する．メソッド名が同じでもノードの始点と終点が異なる場合は別のエッジとして扱う．

更に，Partial Model を構成している元々の設計候補の状態遷移図を命題論理式によって表現する．表現された命題論理式は破線で表されたエッジやノードについて各候補においてどのエッジやノードが使われているかを表し，それらを OR 演算子で結ぶことで表現が可能である．

Partial Model は論理式で表現されたプロパティとの充足可能性を解析することができるとしている．Partial Model Φ_M 上で，プロパティ Φ_P を検査するためには $\Phi_M \wedge \Phi_P$ 及び，$\Phi_M \wedge \neg\Phi_P$ について SAT（充足可能性問題）ソルバで解析を行い，充足可能性問題を解く．

SAT ソルバによる結果に応じて表 1 のようなプロパティ

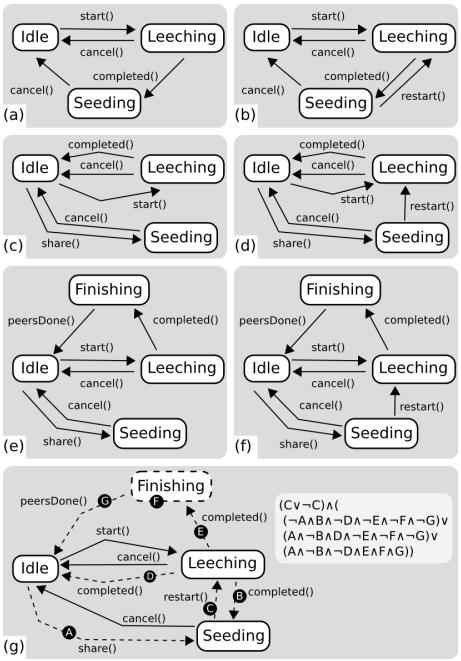

図 3　P2P ファイル共有システムにおける 6 つの設計候補 (a-f) とその候補を統合した Partial Model (g)[2]

表 1　Partial Model M 上でのプロパティ p の検査表 [2]

$\Phi_M \wedge \Phi_P$	$\Phi_M \wedge \neg\Phi_P$	Property p
SAT	SAT	Maybe
SAT	UNSAT	True
UNSAT	SAT	False
UNSAT	UNSAT	(error)

の検査の結果を出すことができる．ここでの True，False はプロパティが充足，非充足に対応している．Maybe については $\Phi_M \wedge \Phi_P$ 及び，$\Phi_M \wedge \neg\Phi_P$ がいずれも SAT のときに出力される．これはある状態遷移図群では $\Phi_M \wedge \Phi_P$ で充足可能であるが，前者以外の状態遷移図群において $\Phi_M \wedge \neg\Phi_P$ が充足可能であるという場合に起こりうる．つまり，プロパティが成り立つかどうかはどのモデルを最終的に選択するかによって異なるため，結果的にプロパティが成り立つかどうかは「不確か」であるため，Maybe と表現されている．いずれも UNSAT である場合はそもそものPartial Model を構成している状態遷移図群が充足不能であると考えられるため，設計自体を見直す必要がある．そのため，このような場合は error として表現をされている．

3.2　不確かさを包容したインターフェース機構 *Archface-U*

3.2.1　*Archface-U* の概要

本開発プロセスでは，*Archface* [6] インターフェース機構を拡張した *Archface-U* [7] インターフェース機構を用いる．

Archface は，アーキテクチャ設計と Java による実装の間のギャップを埋めるためのインターフェース機構である．*Archface* はプログラミングにおけるインターフェースであり，プログラムを実装する際，*Archface* 中に記述されたアーキテクチャの制約に従うことが強制される．*Archface* の開発環境である *iArch* はソースコードを解析し，アーキテクチャ制約に違反していないかを検査する．この検査を以後，型検査と呼称する．もし，この制約に違反した場合はコンパイルエラーとして制約違反を開発者に知らせることで確実にアーキテクチャ設計に従った開発を行うことができる．

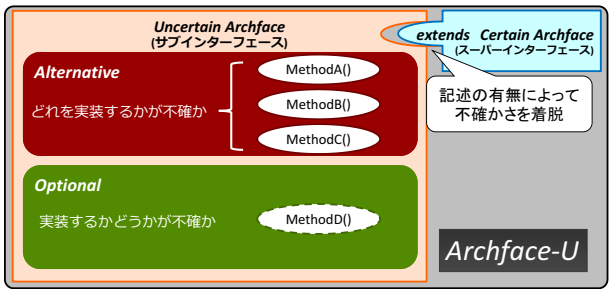

図 4　*Archface-U* の構造

本研究では上記の *Archface* を拡張し，不確かさを包容したインターフェースとして *Archface-U* を定義した．*Archface-U* は従来の *Archface* である *Certain Archface* と，そのサブインタフェースである不確かさを含んだ *Uncertain Archface* の 2 種類のインターフェースによって構成される (図 4)．具体的には，*Uncertain Archface* に *Certain Archface* を記述するか否かによって不確かさを着脱する．

Archface-U は Component-and-Connector アーキテクチャ [1] を記述支援対象としている．このアーキテクチャは計算処理を行うコンポーネント群とそれらの接続関係によってアーキテクチャを記述する．そのため，*Archface-U* はコンポーネントと，そのコンポーネント間の接続関係や協調関係であるコネクターをインターフェースに表現する．

Archface-U では，以下の 2 つの種類の不確かさをインターフェースに記述することが可能である．

(1) ある目的に対していくつかコンポーネントの候補があり，その中でどれを実際にシステムに組み込むかわからないという不確かさ
(2) あるコンポーネントについて実際にシステムに組み込まれるか分からないという不確かさ

この 2 つの不確かさのうち，(1) を *Alternative*，(2) を *Optional* と以下呼称する．

Archface-U では，Java コードのメソッドの呼び出しや実行，クラスやメソッドの宣言などをプログラム点として定義する．コード上のプログラム点と *Archface-U* 上のプログラム点を比較することによってアーキテクチャ設計違反を検査，すなわち型検査を行うことができる．

3.2.2　*Certain Archface* のインターフェース記述

Certain Archface（従来の *Archface*）はコンポーネントとコネクターの 2 つのインターフェースがある．図 3 の P2P ファイル共有システムにおけるインターフェース記述

```
1   interface component Node{
2     void start();
3     void cancel();
4     void completed();
5   }
6   uncertain component uNode extends Node{
7     [void restart();]
8     {
9       void share(),
10      void share(File file)
11    };
12  }
13
14  interface connector cP2PSystem{
15    Node = (Node.start -> Node.cancel -> Node);
16  }
17  uncertain connector ucP2PSystem
18  extends cP2PSystem{
19    Node = (Node.start -> Node.completed
20     -> [Node.restart] -> Node.cancel -> Node);
21  }
```

図 5　P2P ファイル共有システムの *Archface-U* によるインターフェース記述

の例を，図 5 に示す．

コンポーネントインターフェース（interface component）は，実装において宣言するメソッド名を Java インターフェースに準拠した構文によって記述する．コンポーネント名が Java の実装におけるクラス名に対応し，その中にメソッド名を記述することでプログラム点を定義する．図 5 の例では，例えば，start がコード内に宣言されていない場合は制約違反となる．

コネクターインターフェース（interface connector）は，定義されたコンポーネント同士の接続方法を規定し，公開されたプログラム点群をどのように実行し協調させるかを記述する．具体的には，前述のコンポーネントインターフェースによって定義されたコンポーネント同士がどのように接続しているかを FSP（Finite State Process）[4] に準拠した構文によって記述する．図 5 の例では，例えば，Node.start から Node.cancel への接続がない場合コネクターインターフェースの Node における制約違反となる．

3.2.3　*Uncertain Archface* のインターフェース記述

Uncertain Archface は，*Optional*，*Alternative* の不確かさを記述することができる．この 2 種類の不確かさを表現するために 2 つの言語要素を導入する．*Optional* には [ ]，*Alternative* には {} を使い，*Uncertain Archface* のメソッドを囲うことで不確かさを表現する．*Uncertain Archface* ではコンポーネント，コネクター内に上記の不確かさを表す言語要素によって囲まれたメソッドを記述する．また，*Uncertain Archface* は *Certain Archface* のサブインターフェースとして記述する．この継承関係により，不確かさの着脱もスーパーインタフェースの記述の有無によって可能となる．*Uncertain Archface* のスーパーインターフェースは Java の継承ルールに則り，1 つのみとする．

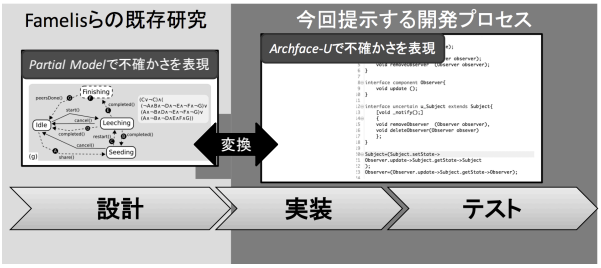

図 6　不確かさを包容した開発プロセス

Uncertain Archface は複数のアーキテクチャ設計を同時に表現をする．例えば，図 5 の uncertain connector 内の Node は，restart が *Optional* であるため，start -> completed -> restart -> cancel と start -> completed -> cancel といった 2 つの接続関係を同時に表現している．

図 5 のコネクターインターフェースは，図 3 のモデル (a) と (b) の関係を簡易的に表現している．このように，Partial Model と *Archface-U* は互いに変換することが可能である．先行研究 [7] では，Partial Model と *Archface-U* の双方向変換のためのアルゴリズムを提示している．

3.3　ソフトウェア開発における不確かさを包容した開発プロセス

我々が提案する開発プロセスでは，不確かさが発生した開発工程によって，異なる方法で不確かさを表現する (図 6)．

まず，設計段階で不確かさが発生した場合，Partial Model を用いて不確かさを表現する．Partial Model を用いて不確かさを表現することで，3.1 節で述べたプロパティの検査を行うことができる．プロパティ検査を行うことによって，不確かさが発生している状況でもシステムの整合性を保ちながら設計を行うことが可能になる．このとき，設計で不確かさが発生し，設計で不確かさが解決する場合は 3.1 節で説明した既存研究 [2] に基づくため，我々の今回提案する開発プロセスには含まれない．

我々の開発プロセスに含まれるケースは，設計で発生した不確かさを表現した Partial Model を *Archface-U* に変換し，実装以降で不確かさを解決する場合，あるいは，実装以降で発生した不確かさを Partial Model へ反映させる場合である．次に，実装，あるいはテスト段階で不確かさが発生した場合，*Archface-U* を用いて不確かさを表現する．*Archface-U* を用いて不確かさを表現することで，3.2 節で述べた型検査を行うことができる．型検査により，不確かさが発生している状況でも，アーキテクチャ設計に沿って

図 7　不確かさを包容した開発支援環境 *iArch-U* の構成

実装を行っているかを確認することができる．

Archface-U では，3.2 節のように，不確かなメソッドを *Optional* メソッドや *Alternative* メソッドを使って表す．不確かさが発生した場合，これらのメソッドを記述し，解決した場合，削除することで，コード上の不確かさの有無を表現することが可能である．

4. 不確かさを包容した開発支援環境 *iArch-U*

不確かさを包容した開発プロセスを支援するために，我々は，*Archface* の開発環境である *iArch* を拡張し，*iArch-U* という開発支援ツールを考案した．開発ツール *iArch-U* は *Archface-U* のコンパイラとそのエディタ，不確かさを含んだ単体テスト支援環境，そして設計におけるプロパティ検査を行うための LTSA (Labelled Transition System Analyser) [4] によって構成される (図 4)．以下では，それぞれについて簡単に説明する．

4.1 *Archface-U* コンパイラ，エディタ

Archface-U エディタは，統合開発環境 Eclipse 上で，*Archface-U* の記述支援を行う．また，*Archface-U* は Java コードのコンパイルが行われると同時に *Archface-U* コンパイラによってコンパイルされ，3.2 節における型検査を Java コードに対して行う．型検査違反がある場合はそれをコンパイルエラーとして返す．なお，*Archface-U* コンパイラとエディタは Eclipse プラグインとして XText とその API，およびソースコードの AST(Abstract Syntax Tree) 解析によって実現している (図 8)．

4.2 不確かさを含んだ単体テスト支援環境

この支援環境は，主に不確かさを含んだテスト駆動開発におけるプロセス支援を行うためのツールである [3]．具体的には，*Archface-U* に記述されたコンポーネント群の中で不確かなものから，一時的にテストに組み込みたい，あるいは組み込みたくないメソッドを選択する．その後，JUnit による単体テストが実行されると選択に応じた AspectJ ソースコードが自動生成される．この AspectJ ソースコードによって既存のテストケースの実行を Weaving し，ユーザの選択を反映させることができる．その結果，ユーザはコンポーネントの選択のみでテストケースに直接手を加えずに安全にテストケースを変更して結果を確認し，不確かさの解決に役立てることが可能となる．

4.3 LTSA

LTSA は Magee らによって開発されたモデル検査ツールである [4]．LTSA 内では，LTS（Labelled Transition System）を FSP によって記述でき，時相論理式を用いてプロパティを記述，および検査をすることができる．

3.2 節でも述べたとおり，*Archface-U* のコネクターインターフェースは FSP をベースにしているため，*Optional* や *Alternative* といった言語要素を除けば，ほぼそのままこのインターフェースを LTSA へ記述をすることが可能である．つまり，LTSA を使うことで実装上に不確かさが含まれていても *Archface-U* を用いてプロパティの検査を行うことができる．

5. ソフトウェア開発における不確かさの発生と解決

本章では，ソフトウェア開発において様々な場面で発生する不確かさに対し，どのように解決するかをその開発プロセスの観点から説明し，我々のアプローチの有用性を示す．以降では，図 3 の P2P ファイル共有システムの設計候補における不確かさをこの開発プロセスを用いて解決するシナリオを示す．これにより，本開発プロセスを用いることで具体的にどのようなメリットを得ることができるのかを示す．

5.1 不確かさが設計段階で発生，実装段階で解決

この節では，設計段階で不確かさが発生し，実装段階でその不確かさが解決されるシナリオを示す．

5.1.1 不確かさへの従来の対処

P2P ファイル共有システムの例において，現在，顧客の要求が不明確であり，図 3 のモデル (a) と (b) のいずれにするかが決まっていないとする．モデル (a) と (b) の違いは，状態 `Seeding` において `restart` を行うかどうかである．

DevOps (開発と運用を同時に行う開発手法) では，新しい要求が絶えず生まれたり消えたりするため，このような状況が発生しやすいと考えられる．DevOps では，設計が確定をしない状態で実装を行わなくてはならない状況が多々ある．

しかし，従来，このような不確かさを設計に記述するような適切な記述法は存在しなかった．そのため，モデルや仕様書に形式的でない方法で記述を行っていた．また，今回の例のモデル (a) と (b) のように，設計候補がいくつかある場合，実装プロセスにおいて現在どの設計候補についての実装を行っているかどうか確認することができない．最悪の場合，設計候補が分かっているにもかかわらず，そのどの設計候補にも従っていない実装を行ってしまうこと

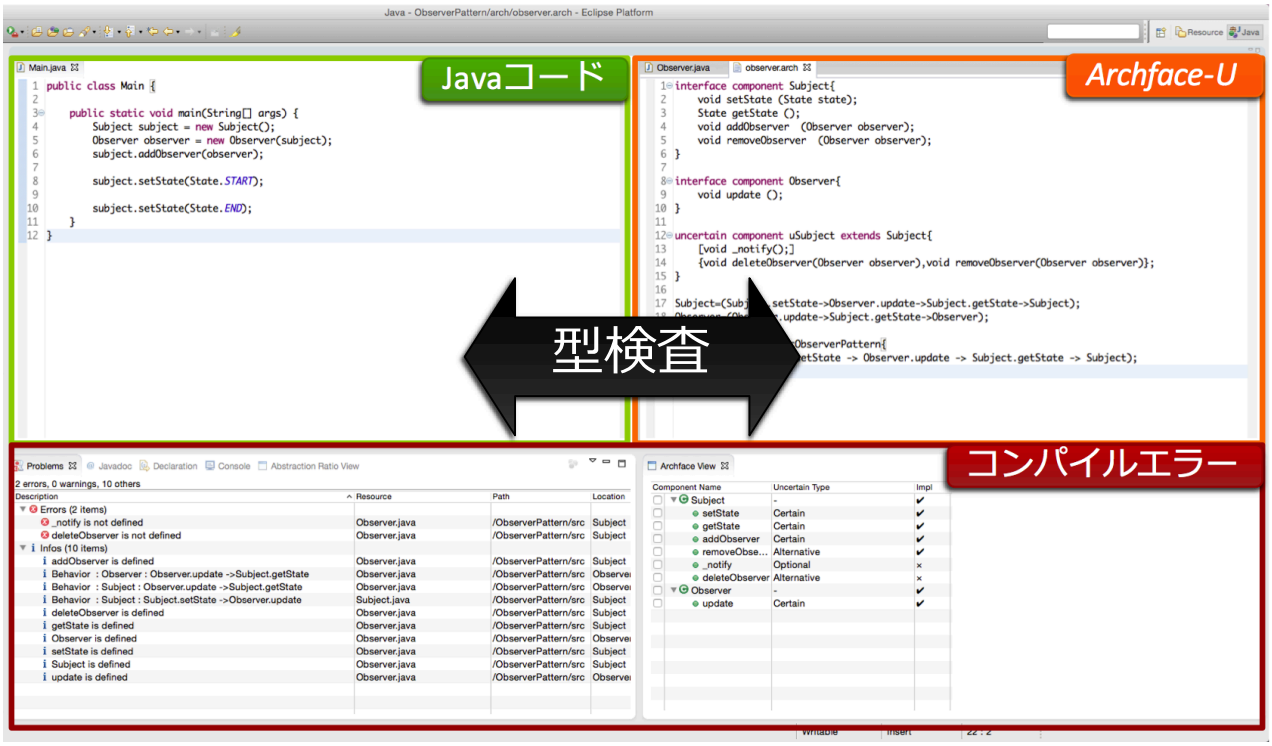

図 8　*Archface-U* コンパイラ，エディタ

もあり得る．

5.1.2　我々の開発プロセスによる不確かさへの対処

不確かさの表現: まず，設計段階で発生した「`restart` を行うかどうか」という *Optional* な不確かさを表すことを考える．我々のアプローチでは，この *Optional* な不確かさを，設計段階では Partial Model を使って表すことができる．3.1 節にも示したとおり，Partial Model を作成することで，このフロー自体のプロパティ [*1] について検査を行うことができ，顧客が提示する不確かな要求がフローの前提条件を崩さないかを確認することができる．

次に，不確かさを包容しつつも安全性が確認できた設計モデルを，実装に落としこむことを考える．このとき，本開発プロセスでは Partial Model を *Archface-U* へ変換する．*Archface-U* へ変換することで，Partial Model で表現された不確かさを引き継いだまま実装へつなげることができる．

Archface-U は単なるドキュメントとして不確かさを実装へ引き継ぐだけではなく，3.2 節でも述べたように，*Archface-U* は実装に対して型検査を行う．型検査をこの例に用いた場合，現在，モデル (a)，(b) のいずれに従って実装を行っているかを確認することが可能となる (図 9)．これは，4.1 節で述べたコンパイラによるメッセージによって確認をすることができる．また，モデル (a)，(b) のいずれにも従っていない実装を行っている場合，その結果がコンパイルエラーとして表示される．そのため，不確かさを抱えている場合でも設計に従った実装を行うことが可能となる．

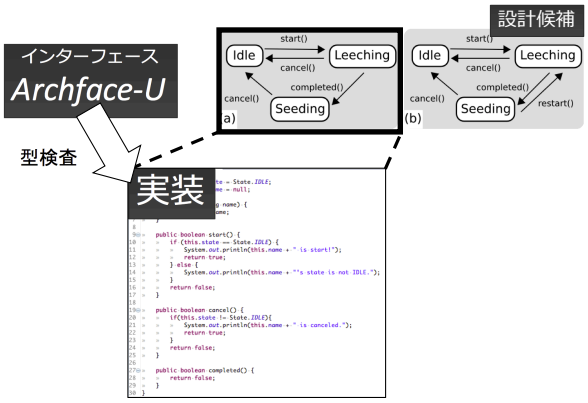

図 9　*Archface-U* の型検査による実装が従っているモデルの判別

不確かさの解決: 不確かさが実装段階で解決される場合を考える．今回の例では，実装段階で顧客がモデル (a) の設計でシステムを作って欲しいと要求が確定したとする．このことで，不確かさが解決したため，*Archface-U* の記述を変更することで，不確かさが解決したことを表す．具体的には，*Archface-U* において，`restart` の記述を削除するだけでよい．逆に，顧客がモデル (b) を望んだ場合は `restart` の [ ] を外すことによって確定することができる．

5.2　不確かさが実装段階で発生，実装段階で解決

この節では，実装段階で不確かさが発生し，実装段階で

[*1] 例えば，今回の例では「どの状態においても `Idle` 状態へ戻ることができる」などが考えられる．

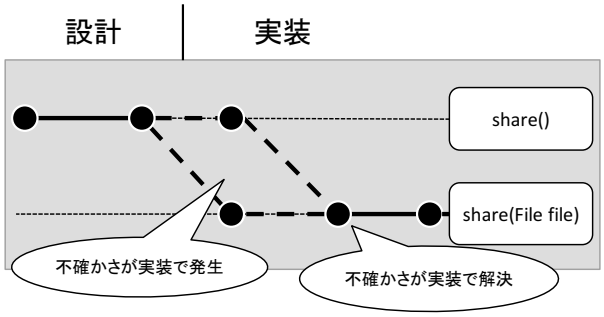

図 10　*Archface-U* と VCS を用いた不確かさの追跡

その不確かさが解決されるシナリオを示す．

5.2.1　不確かさへの従来の対処

P2P ファイル共有システムの例における `share` メソッドに着目をする．`share` メソッドは，他ノードにファイルの共有を開始するメソッドである．実装者はこの `share` メソッドを実装している最中，`share` の引数に File オブジェクトを取るべきと考えた．しかし，設計モデルでは引数についての言及がないため，引数有りと引数無しのどちらで実装をするべきかが現状では分からないとする．

このような状態の場合，引数があるかどうかが未確定な状況を書く適切な方法がない．そのため，実装者は引数を使いたいため，引数有りで定義を行い，設計との相違があることをコメント等で残すことになる．しかし，このような方法は設計に引数がある旨を書き忘れがちである．さらに，このような不確かな状態が実装上に多数あった場合，その管理をコメントだけで行うのは難しい．

5.2.2　我々の開発プロセスによる不確かさへの対処

不確かさの表現: 我々のアプローチでは，このような場合，*Archface-U* 上に，`share` メソッドの引数の有無について，*Alternative* な不確かさとして記述をする．*Archface-U* に記述することで，アーキテクチャ設計に従った実装を行うことができるだけでなく，初期モデルに引数がなかったことをインターフェースとして残すことができる．

Archface-U と Git などの VCS(版管理システム：Version Controll System) を併用すると，不確かさの履歴を追跡することができる．これは，*Archface-U* はインターフェースであるため，コードの履歴をたどることによってどの段階で不確かさが発生し，解決したかを観察することができるためである．今回の例であれば，初期の設計で引数がなかった `share` メソッドが，実装を始めたときに `share` メソッドの引数の有無について不確かさが生まれたことが VCS の履歴を観察することで分かる (図 10)．

不確かさの解決: 今回の例では，以下の 2 通りの解決方法が考えられる．

(1) 引数が必要なことが分かり，設計へ手戻りして設計を修正することによって不確かさを解決する
(2) 引数が不要なことが分かり，実装のみで不確かさが完結する

このいずれにおいても，*Archface-U* の記述を変更することで解決したことを示すことができる．(1) の場合，設計モデルが確定したと同時に *Archface-U* の記述を変更することで，確実に設計と実装のトレーサビリティを取ることが可能である．(2) の場合，実装のみの不確かさとして処理をすることができるので，実装が確定したら *Archface-U* の記述を変更する．

5.3　不確かさがテスト段階で解決

この節では，テストによって不確かさが解決されるシナリオを示す．

5.3.1　不確かさへの従来の対処

P2P ファイル共有システムの例において，テスト駆動開発を行うことを考える．現在，`share` メソッドの実装が完了しているものとし，それについてのテストケースも作り終えたとする．また，実装した `share` メソッドはこのテストを通過し，問題がない状態であったとする．ここで，顧客が大容量のファイルの送信にも対応するように非機能要求を提示してきたとする．開発者は，この変更によってテストケースが合格できなくなったため，`share` メソッドの実装の改善を行う．また，`share` メソッドは最悪この非機能要求に対応できなかった場合用いられることを考え，新たに `bigDataShare` メソッドを作成するようにした．

この時，`bigDataShare` メソッドのテストには `share` メソッドのテストケースを用いることができる．しかし，どちらのメソッドが最終的に使われるかが定まらない限りは何度もテストケースを書き直すことが想定され，これはコードの保全性から考えると好ましくない．今回の例では，2 つのメソッドの分離であるが，他のメソッドにも同じように非機能要求の追加や更新があることが考えられる．その場合，各々において正確にテストケースを再現することはテストケースの直接変更のみでは難しい．

5.3.2　我々の開発プロセスによる不確かさへの対処

不確かさの表現: 我々のアプローチでは，このような場合，*Archface-U* のコンポーネントインターフェースにおいて `share`，`bigDataShare` を *Alternative* として記述する．この表現により，テストケース内において `share`，`bigDataShare` の 2 種類の呼び出しパターンを簡単に表現することが可能である．

不確かさの解決: このような不確かさの解決を支援するために，我々は *Archface-U* に基づいた AOP (Aspect Oriented Programming) を行うことを提案する．4.2 節で述べたように，我々のツールを用いると，コンポーネントインターフェースの選択によって自動的にテストケースを Weaving する AspectJ ソースコードが生成することができる (図 11)．この方法を用いることによって，テストケースを直接変更する必要がなくなるため，`share` についてのテスト

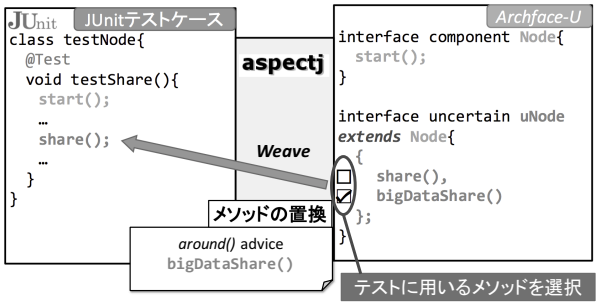

図 **11** *Archface-U* を用いたテスト駆動開発

ケースを維持したまま `bigDataShare` についてのテストを行うことができる．非機能要求を解決することができれば，`share` のテストケースを `bigDataShare` のテストケースへ変更し，*Archface-U* の記述を変更することによって不確かさが解決したことを表現することができる．

6. まとめ

本論文では，*Archface-U* とその支援環境 *iArch-U* を用いた不確かさを包容した開発プロセスを提案した．今回提案したプロセスを用いて，P2P ファイル共有システムの例で発生しうるあらゆる不確かさを表現し，解決するシナリオを提示することができた．さらに，本開発プロセスを適用することで具体的にどのようなメリットを得ることができるのかをシナリオを通じて提示することができた．

以上のことにより，従来より提示してきた不確かさを包容するインターフェース *Archface-U* の有用性を示すことができた．

謝辞　本研究は，文部科学省科学研究補助費基盤研究（A）（課題番号 26240007）による助成を受けた．

参考文献

[1] Allen, R. and Garlan, D.: Formalizing Architectural Connection, *Proceedings of the 16th International Conference on Software Engineering*, pp. 71–80 (1994).
[2] Famelis, M., Salay, R. and Chechik, M.: Partial Models: Towards Modeling and Reasoning with Uncertainty, *Proceedings of the 34th International Conference on Software Engineering*, pp. 573–583 (2012).
[3] Fukamachi, T., Ubayashi, N., Hosoai, S. and Kamei, Y.: Poster: Conquering Uncertainty in Java Programming, *Proceedings of the 37th International Conference on Software Engineering*, pp. 823–824 (2015).
[4] Magee, J. and Kramer, J.: *State Models and Java Programs* (1999).
[5] Perez-Palacin, D. and Mirandola, R.: Uncertainties in the Modeling of Self-adaptive Systems: A Taxonomy and an Example of Availability Evaluation, *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, pp. 3–14 (2014).
[6] Ubayashi, N., Nomura, J. and Tamai, T.: Archface: A Contract Place Where Architectural Design and Code Meet Together, *Proceedings of the 32nd International Conference on Software Engineering*, pp. 75–84 (2010).
[7] 深町拓也，鵜林尚靖，細合晋太郎，亀井靖高：不確かさを包容する Java プログラミング環境，情報処理学会研究報告．ソフトウェア工学研究会報告，Vol. 2015, No. 21, pp. 1–8 (2015).